Deliverable D1.2.1

Algorithms for the analysis and visualization of evolving networks. Analysis of large Webgraphs and evolving network available to the consortium.
Start date of the project: January 2004

Duration: 48 months

Project Coordinator: Prof. Dr. math. Friedhelm Meyer auf der Heide
Heinz Nixdorf Institute, University of Paderborn, Germany

Due date of deliverable: June 2005
Actual submission date: July 2005
Dissemination level: PU – public

Work Package 1.2: Monitoring and Measuring the Evolving Web

Participants: Università Degli Studi di Roma “La Sapienza” (UDRLS), Italy
Universität Paderborn (UPB), Germany
Universität Karlsruhe (UniKar), Germany
Université de Paris-Sud XI (UPS), France
Università Degli Studi di Milano (UniMI), Italy

Authors of deliverable: Stefano Leonardi (Stefano.Leonardi@dis.uniroma1.it)
Luciana S. Buriol (buriol@dis.uniroma1.it)
Debora Donato (donato@dis.uniroma1.it)
Marco Gaertler (gaertler@ira.uka.de)
Christian Sohler (csohler@upb.de)
Gereon Frahling (frahling@upb.de)
1 Introduction

One of the main goals of DELIS SP1 is to develop algorithms and tools for monitoring and discovering the principal properties of evolving networks. This ambitious task is accomplished by exploring different research directions. A first research line is constituted by a fine-grain analysis of the details of the Web graph with the goal of discovering its inner topological and statistical properties. This kind of analysis can be seen as a starting point for a systematic study of the temporal evolution of the complex networks. This study is mainly performed considering sequential snapshots captured over the time. Managing these kind of data requires a big amount of resources especially in terms of memory storage. An alternative approach consists in developing data stream algorithms for the analysis of massive graph on the fly, like the Web, as they are collected. In particular, within DELIS SP1, the focus has been on problems like maintaining degree distribution and identifying clustering. The study of large evolving network can be significantly improved by the means of visualization techniques as graphical representations have the great advantage to improve the human perception. So far the following results have been achieved on the research issues outlined above:

- A detailed study of the inner structure of the bow-tie has been conducted at URLS. Four different crawls have been extensively measured revealing that although the global samples present the same macroscopic statistics previously reported in literature, the single components, CORE IN and OUT, have a substantially different structure.
- A previously known data stream algorithm has been used for computing the indegree rarity distribution of the graph over the arc arrival. It has been shown experimentally that the results are very close to the optimal even when a low precision is requested. The same algorithm has been used also for estimating the density of small bipartite cliques.
- A general technique to approximately solve a number of fundamental clustering problems over dynamic geometric data streams has been developed at UDP. The main contribution consists in a general method to construct a coreset by combining statistics about the distributions of points in a certain number of nested grids. It is remarkable that such an algorithm have applications in a distributed scenario as, for example, one in which the input data is distributed over a number of data stream with different computers monitoring each of them.
- An integral approach for static visualizations of various type of networks has been developed at UniKarl. These layouts are very flexible and can incorporate divers analytic properties. In addition to respecting esthetic criteria such as crossing minimization and compactness, the readability and the human perception of the included analytic characteristics are improved. Based on these fundamental results, different methods to transfer knowledge of the static case to the dynamic case have been explored.

2 Results

2.1 Mining the inner structure of the Web graph

2.1.1 State of the art

In the past decade the world has witnessed the explosion of the World Wide Web from an information repository of a few millions of hyperlinked documents into a massive world-wide “organism” that serves informational, transactional, and communication needs of people all over the globe. Naturally, the Web has attracted the interest of the scientific community, and it has been the subject of intensive research work in various disciplines.
Table 1: Sizes and bow-tie components for the different crawls and the Alta Vista graph

<table>
<thead>
<tr>
<th></th>
<th>Italy</th>
<th>Indochina</th>
<th>UK</th>
<th>WebBase</th>
<th>AltaVista</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>41.3M</td>
<td>7.4M</td>
<td>18.5M</td>
<td>135.7M</td>
<td>203.5M</td>
</tr>
<tr>
<td>edges</td>
<td>1.15G</td>
<td>194.1M</td>
<td>298.1M</td>
<td>1.18G</td>
<td>1.46G</td>
</tr>
<tr>
<td>CORE</td>
<td>29.8M (72.3%)</td>
<td>3.8M (51.4%)</td>
<td>1.2M (65.3%)</td>
<td>44.7M (32.9%)</td>
<td>56.4 (27.7%)</td>
</tr>
<tr>
<td>IN</td>
<td>13.8K (0.03%)</td>
<td>48.5K (0.66%)</td>
<td>312.6K (1.7%)</td>
<td>14.4M (10.6%)</td>
<td>43.3 (21.3%)</td>
</tr>
<tr>
<td>OUT</td>
<td>11.4M (27.6%)</td>
<td>3.4M (45.9%)</td>
<td>5.9M (31.8%)</td>
<td>53.3M (39.3%)</td>
<td>43.1 (21.2%)</td>
</tr>
<tr>
<td>TENDRILS</td>
<td>6.4K (0.01%)</td>
<td>50.4K (0.66%)</td>
<td>139.4K (0.8%)</td>
<td>17.1M (12.6%)</td>
<td>43.8 (21.5%)</td>
</tr>
<tr>
<td>DISC</td>
<td>1.25K (0%)</td>
<td>101.1K (1.4%)</td>
<td>80.2K (0.4%)</td>
<td>6.2M (4.6%)</td>
<td>16.7 (8.2%)</td>
</tr>
</tbody>
</table>

The first large-scale study of the Web graph was performed by Broder et al. [10] and it revealed that the Web graph contains a giant component that consists of three distinct components of almost equal size: the CORE, made up of a single strongly connected component; the IN set, comprised by nodes that can reach the CORE but cannot be reached by it; the OUT set, consisting of nodes that can be reached by the CORE but cannot reach it. These three components form the well known bow-tie structure of the Web graph.

The bow-tie picture describes the macroscopic structure of the Web. However, very little is known about the inner structure of the components that comprise it. Broder et al. [10] pose it as an open problem to study further the structure of those components. Dill et al. [2] demonstrated that the Web exhibits self-similarity when considering “Thematically Unified Clusters” (TUCs), that is, sets of pages that are brought together due to some common trait. Thus the Web graph can be viewed as the outcome of a number of similar and independent stochastic processes. Pennock et al. [4] also argue that the Web is the sum of stochastic independent processes that share a common (fractal) structure.

The findings about the structure of the Web generated a flurry of research in the field of random graphs. Given that the standard graph theoretic model of Erdős and Rény [3] is not sufficient to capture the generation of the Web graph, various stochastic models were proposed [9, 4, 6]. Most of them address the fact that the in-degrees must follow a power-law distribution [9]. The copying model [6] generates graphs with multiple bipartite cliques [11].

### 2.1.2 Main Contributions

We experiment with four different crawls. The first three crawls are samples from the Italian Web (the .it domain), the Indochina Web (the .vn, .kh, .la, .mm, and .th domains), and the UK Web (the .uk domain) collected by the "Language Observatory Project" 1 and the "Istituto di Informatica e Telematica" 2 using UbiCrawler [5]. The fourth crawl is a sample of the whole Web, collected by the WebBase project at Stanford 3 in 2001. This sample contains 360 millions of nodes and 1.5 billion of edges. In order to eliminate non-significant data, we pruned the frontier nodes (i.e. the nodes with in-degree 1 and out-degree 0, on which the crawler has been arrested). The sizes of the crawls are shown in Table 1.

**Macroscopic measurements**  We observe the same macroscopic properties previously reported in the literature: the degree distributions follow a power-law, and the graph has a bow-tie structure, although (depending on the crawler) a little different in shape.

---

1[www.language-observatory.org](http://www.language-observatory.org)
2[www.itt.cnr.it](http://www.itt.cnr.it)
As a first step in the understanding of the individual components we compute the same macroscopic measures as for the whole Web graph. We compute the in-degree, out-degree and SCC size distributions for each of the IN, OUT, TENDRILS and DISC graphs. It is obvious that the same macroscopic laws that are observed on the whole graph are also present in the individual components.

**The inner structure of the bow-tie** We study in detail the inner structure of the bow-tie graph. We perform a series of measurements on the CORE, IN and OUT components. Our measurements reveal a number of surprising fact. Although the individual components share the same macroscopic statistics with the whole Web graph, they have substantially different structure. The first indication that the self-similarity conjecture is not true comes from the fact that there is no large SCC in the IN and OUT components. For the OUT component, in all crawls, the largest SCC is only a few thousands of nodes. Given that the size of the OUT component is in the order of millions, the largest SCC is staggeringly small. We observe a similar phenomenon for the IN component. Therefore, it appears that there exists no sizable SCC in the IN and OUT components that could play the role of the CORE in a potential bow-tie.

Moreover we discovered that there is no giant WCC in either of the two components. In fact, there is a large number of WCCs per component and their sizes follow a power law distribution. The largest WCC is never more than 30% of the component it belongs to, which is small compared to the giant WCC in the Web graph, which contains more than 90% of the nodes. We also investigate how the nodes in the largest WCCs in the IN and OUT components are connected to see if they organized in a bow-tie shape. Our investigation revealed that starting from the largest SCC in the WCC, we can create a bow-tie that is no more than 15% of the WCC (for the Italian Web), and usually less than 5%. The rest belongs to the DISC component. In order to better understand how the nodes in IN and OUT are arranged with respect to the CORE, we split the nodes in the IN and OUT components in *levels* depending on their distance from the CORE. In all graphs, the depths of the components are relatively small. Furthermore, most nodes are concentrated close to the CORE. Typically, about 80-90% of the nodes in the OUT component are found within the first 5 layers. Therefore, we conclude that the IN and OUT components are shallow and highly fragmented. They are comprised of several sparse weakly connected components of low depth. Most of their volume consists of nodes that are directly linked to the CORE.

We study also the structure of the CORE and, as first step, we examine its relation with the IN and OUT components. We define an *entry point* to the CORE to be a node that is pointed to by at least one node in the IN component, and an *exit point* to be a node that points to at least one node in the OUT component. A *bridge* is a node that is both an entry and an exit point. We also compute the in-degree distribution of the entry points when we restrict the source of the links to be in the IN component, and, as expected, we observe a power law. This implies that most nodes “serve” as entry points to just a few nodes in the IN component, while there exist a few nodes that serve as entry points to a large number of IN nodes. Similar distributions are obtained when we consider the out-degree distribution of the exit points, restricted to the OUT component.

We then study the connectivity of the CORE. We first look for nodes that are loosely connected to the CORE. We define a *connector* to be a node of the CORE that has a single in-coming and out-going link. A connector forms a *petal* if the source of the incoming link, and the target of the out-going link are the same node. Large number of connectors would imply weak connectivity of the CORE. The number of connectors is on average around 5%. Of these 20 to 45% are petals. Therefore, connectors are only a small part of the CORE.

In order to further understand the connectivity of the CORE, we test the resilience of the CORE to targeted attacks by performing the following experiment. For some $k$ we delete all nodes from the CORE that have total degree (in-degree plus out-degree) at least $k$. We then compute the size of the largest SCC in the resulting graph. We observe that the threshold on the total degree must become
as low as 100 in order to obtain an SCC of size less than 50% of the CORE.

All the experiments performed suggest a refinement of the bow-tie picture the daisy structure of the Web graph, that takes our findings into account.

2.2 Using data streams algorithms for computing properties of large graphs

2.2.1 State of the art

Data stream algorithms aim to maintain the underlying information of a stream of data, using small memory space. The data is processed on the fly, as it is generated, or it can also be read from second memory devices. Typical applications of data stream algorithms are originated from massive datasets such as network traffic measurements, telephone call records, biological datasets and atmospheric observations. In these applications is unnecessary or impractical to read data multiple times. In many cases, the data is not even stored. This work focuses on a “new” natural application for data streams. We are interested in using data stream algorithms for monitoring statistical and topological properties of large graphs such as the webgraph. The graph read in a streaming fashion considers each edge as an item and the stream is not required to be structured.

The main advantage of using data streams instead of exact algorithms is that the space used for managing and mining the stream is small, without resorting to external memory algorithms. Furthermore, results can be output anytime during the stream processing, not requiring that the whole data input be processed in advance. On the other hand, data stream algorithms do not provide exact values, but an approximation that depends on the precision required and the amount of resources we are willing to invest.

Several theoretical results have been proposed in this new research field, some of them have not yet been implemented and experimented, some of them are not practical. We use the algorithm proposed by Datar and Muthukrishnan [8] for estimating the $\alpha$-rarity of a stream that is the measure of number of items that repeat exactly $\alpha$ times in the stream. The $\alpha$-rarity $\rho_\alpha$ is defined as the ratio $\rho_\alpha = \frac{\#\alpha\text{-}rare}{\#\text{distinct}}$, where $\#\alpha\text{-}rare$ is the number of elements that appear exactly $\alpha$ times in the stream and $\#\text{distinct}$ denotes the number of distinct items in the stream.

2.2.2 Main Contributions

We observe how a data stream algorithm behaves in practice for computing the indegree rarity distribution of a graph over the arc arrivals. More specifically, we maintain the distribution of the number of nodes that has a given indegree over the total number of different nodes seen in the stream so far. We show experimentally that the results are very close to the optima even when a low precision
is requested. The original algorithm proposes the use of min-wise hash functions, whereas we use universal hashing [8]. This decision is due to the fact that computing min-wise hashing consumes about two orders of magnitude more time than universal hashing without providing better results in practice for the graphs we have tested.

When considering a specific structure in the data stream, other properties can be computed. For example, reading the stream in an adjacency list fashion, the same rarity algorithm can be used for estimating the density of minors such as small bipartite cliques.

We conducted our experiments on streams of Wikipedia graphs. A graph of this type is generated from the link structure of the online and free-content encyclopedia Wikipedia (www.wikipedia.org).

We generate streams of edges of the Wikipedia graphs following their generation on time. In our experiments we use the graphs wikiEN, wikiDE, wikiFR, wikiIT, wikiPT from the datasets extracted from the English, German, French, Italian and Portuguese languages, respectively. The graphs were obtained from an old dump of July 2004. Due to space restrictions, we limited the presentation of experimental results in this extended abstract to the wikiEN and wikiPT graphs. Some comments are added about the experimental results on the other three graphs. Graph wikiPT contains 8,131 nodes and 48,168 edges, while graph wikiEN is two orders of magnitude larger containing 286,754 nodes and 4,065,530 edges.

The algorithms were coded in g++ version 3.3.2. The experiments were conducted in an Intel Pentium IV, with 1GB RAM, running Mandrake 9.0. Due to the excessive computational time spent by min-wise hash functions, we use universal hash functions instead. We optimized an online available implementation of min-wise hash functions, but still it takes too long. We used the hash function (hash31) and the random number generator (prng.int) from the online available codes from the MassDAL group of Rutgers (http://www.cs.rutgers.edu/muthu/massdal-code-index.html). The graph is read as a sequence of arcs. The endpoint of each edge is hashed for the rarity indegree hashing, and triples of nodes are hashed for counting the number of bipartite cliques of size three.

For a good approximation, a larger number of hash functions are required. For example, if we set $\epsilon = p = \tau = 0.1$, 10,000 hash function are required. For $\epsilon = p = \tau = 0.2$, just 437 hash functions are needed. But we observed, that even with a small number of hash functions, the results are close to the optima. Figure 2 presents results when using only 100 and 1000 hash functions. The lines are plot for a logarithmic number of indegree values. The plot omits results for indegree higher than 63 for the sake of clarity of the figure, but a complete plot would present additional lines on the bottom of the figure, appearing on increasing order of the number of edges processed.

For the windowed case, similar quality of results were found. Figure 3 presents results for windows of 100,000 items, estimated using 100 hash functions.

We also found good approximation when using the $\alpha$-rarity algorithm for computing the rarity distribution of $k(i,3)$ on the graph. Results for $i=1,2,3$ are plot in Figure 4. The plot is in log scale to be able to visualize all three distributions clearly on the same plot. Usually the number of $k(1,3) \gg k(2,3) \gg k(3,3)$. The difference between this values decrease with the increase of $i$. Observe, for example, the precision on results between the estimated and exact computation of $k(1,3)$ and $k(2,3)$. Since $k(1,3)$ is found many more times than $k(2,3)$, the results are more accurate. For values of $i > 4$ we did not plot for the sake of clarity of the plot, but the precision on the results decrease with the increase of $i$. As expected, we have less precision for computing $\hat{\rho}$ of $\alpha$-rare elements that occur less frequently.

We finalize the experimental results section with a time analysis as a function of the number of hash functions used and the number of elements hashed. Table 2.2.2 presents the average time spent for computing windows of 1,000, 10,000, 100,000 and 1,000,000 items ($\tilde{W}$). For each value of $\tilde{W}$, the use of 100 and 1000 universal hash functions are considered. The first column ($\tilde{W}$) indicates the number...
Figure 2: Estimated and exact indegree rarity distribution computed for edges arrivals of graph wikiEN. The estimation makes use of 1000 (graph on the left) and 100 (graph on the right) universal hashing functions. Values are presented to $\alpha$ up to 63, presented as $\log_2$ plot. This plot presents the percentage of nodes with a given indegree (y-basis) considering the amount of edges processed so far (x-basis). Results are plot every 100,000 items processed.

Figure 3: Estimated and exact indegree rarity computed for edges arrivals of graph wikiEN under the windowed model. Windows are comprised of 100,000 items and 100 universal hashing functions were used. Values are presented to $\alpha$ up to 7, presented as $\log_2$ plot.

Figure 4: Plot in log scale of the estimated and exact $k_i,3$ rarity distribution, for $i=1,2$ and 3, computed for edges arrivals of graph wikiPT. The estimation makes use of 1000 universal hash functions. This plot presents the percentage of triples pointed by exactly $i$ nodes (y-basis) considering the amount of triples seen so far (x-basis). The triples are computed accordingly with the edges arrivals. Results are plot every 10,000 triples processed.
of elements processed for the respective time information presented. It also indicates the size of the windows for the windowed case. The times are an average of the overall times of all windows of \( W \) items processed. Times are about constant over the windows. The second column, \#h, indicates the number of universal hash functions used. The last three columns, \( I \), \( W \) and \( k(i,3) \), presents the average time for the indegree rarity distribution, windowed indegree rarity distribution and \( k(i,3) \) rarity distribution, respectively. For the \( k(i,3) \) case, a second time value is printed, indicating the time spent to process \( W \) triples. Similar results for approximation and times are observed for the other 4 graphs (we omit results due to space limitations). For the \( k(i,3) \) case, values for \( W \) of triples processed were add in parenthesis. Computational times using the optimized min-wise hash function was omitted due to the excessive time spent. For example, for computing 10,000 items and using 100 hash functions, for the indegree rarity distribution, the algorithm takes on average more than 200 seconds, whereas only 0.03 s. is spent using universal hashing. Before the optimization, it was spending thousands of seconds for the same configuration.

<table>
<thead>
<tr>
<th>( W )</th>
<th>#h</th>
<th>( I )</th>
<th>( W )</th>
<th>( k(i,3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1,000</td>
<td>0.003</td>
<td>0.03</td>
<td>9.36 (0.015)</td>
</tr>
<tr>
<td>1000</td>
<td>1,000</td>
<td>0.03</td>
<td>0.67</td>
<td>118.53 (0.19)</td>
</tr>
<tr>
<td>100</td>
<td>10,000</td>
<td>0.03</td>
<td>0.37</td>
<td>131.01 (0.21)</td>
</tr>
<tr>
<td>1000</td>
<td>10,000</td>
<td>0.30</td>
<td>10.22</td>
<td>1166.61 (1.87)</td>
</tr>
<tr>
<td>100</td>
<td>100,000</td>
<td>0.32</td>
<td>3.90</td>
<td>1272.67 (2.04)</td>
</tr>
<tr>
<td>1000</td>
<td>100,000</td>
<td>2.93</td>
<td>141.35</td>
<td>11765.96 (18.86)</td>
</tr>
<tr>
<td>100</td>
<td>1,000,000</td>
<td>3.24</td>
<td>40.54</td>
<td>12776.61 (20.48)</td>
</tr>
<tr>
<td>1000</td>
<td>1,000,000</td>
<td>29.32</td>
<td>1708.30</td>
<td>117859.27 (188.92)</td>
</tr>
</tbody>
</table>

Table 2: Average computation times in seconds for streams of fix size from the wikiEN (results for the indegree rarity distribution) and wikiPT (results for the \( k(i,3) \) rarity distribution) graphs. For each one of the three applications (indegree, windowed indegree and \( k(i,3) \) rarity distribution), times are printed for each \( W \) elements hashed, considering the use of 100 and 1000 hash functions. For the \( k(i,3) \) column, values for \( W \) of triples processed were add in parenthesis.

Times presented for the indegree rarity distribution are very small, even when considering the larger windows. Another observation is that the time grows linearly with the increase of the number of hash functions used and with the number of items considered in each window considered. For the windowed case, more time is spend. That happens because each update on the dynamic lists take \( O(L) \), where \( L \) represents the size of the list. For the \( k(i,3) \) computation, long times are observed. The bottleneck of this application is that all triples of nodes within and adjacency list have to be computed.

### 2.3 Clustering Large Data Sets

#### 2.3.1 State of the art

Clustering is the computational task to partition a given input into subsets of equal characteristics. These subsets are usually called clusters and ideally consist of similar objects that are dissimilar to objects in other clusters. This way one can use clusters as a coarse representation of the data. We loose the accuracy of the original data set but we achieve simplification (this is somewhat comparable to lossy compression). The quality of a partition is rather problem dependent, there is no general clustering algorithm. Consequently, over the years many different clustering algorithms have been developed. These algorithms can be characterized as hierarchical algorithms or partitioning algorithms.
Hierarchical algorithms build a hierarchy of clusters, i.e. every clusters is subdivided into child clusters, which form a partition of their parent cluster. Depending how the hierarchy is built we distinguish between agglomerative (bottom-up) and divisive (top-down) clustering algorithms.

Partitioning algorithm try to compute a clustering directly. For example, they try to compute a clustering by iteratively swapping objects or groups of objects between the clusters or they try to identify dense areas containing many points.

The most prominent and widely used clustering algorithm is Lloyd’s algorithm sometimes also referred to as the k-means algorithm [31, 49, 50]. This algorithm requires the input set to be a set of points in the d-dimensional Euclidean space. Its goal is to find k cluster centers and a partitioning of the points such that the sum of squared distances to the nearest center is minimized.

The main benefit of Lloyd’s algorithm is its simplicity and its foundation on analysis of variances. However, it also has a number of drawbacks. First of all, the computed solution converges to a local minimum [60] and it may be far away from the global one. Also, the quality of the computed solution depends strongly on the initial solution and is rather sensitive to outliers. Another problem is that the number of clusters has to be predetermined by the user. And, finally, the algorithm lacks scalability to large data sets.

We address the latter issue in our work and obtain scalable algorithms not only for the k-means problem but also for a number of related clustering problems including k-median (k-medoid) and MaxCut. We believe that our techniques can be also applied to other related clustering variants including Gaussian expectation maximization, fuzzy k-means clustering and many other problems.

2.3.2 Main Contributions

Our method reduces the input point set to a weighted point set (point weights stand for multiple points). This weighted point set has the property that for every set of k centers the cost for the weighted point set is provably almost the same as the cost of the original point set. Our new point set has size in O(log n) for constant ε and d. We call such a small weighted set of points a coreset.

We present a dynamic data structure (e.g., one supporting insertions and deletions) that maintains in poly(log n) space such a coreset for a sequence of n insertions and deletion of points. Once we have computed such a coreset we can use an arbitrary algorithm (e.g., Lloyd’s algorithm) to obtain a good clustering. The benefit is obvious. We have to run Lloyd’s algorithm on a set of size O(log n) rather than n and hence we obtain a tremendous speedup.

We also develop techniques to maintain a random sample of a dynamically evolving point set. We show that these techniques can be used to solve a number of geometric problems. Random sampling is often used to reduce the size of data sets in the context of clustering. For example, it has been shown that a random sample can be used to obtain a clustering of good quality for the k-median problem (the proofs can be easily carried over to the k-means clustering problem). We also show how to approximate the weight of the Euclidean minimum spanning tree, which is another structure that is commonly used in the context of clustering.

k-Means Clustering using Coresets We develop a general technique to solve a number of fundamental problems over dynamic geometric data streams approximately. Given is a data stream of insert and delete operations of points in a discrete Euclidean space [32]. Each point has integer point coordinates between 1 and Δ. One can look at this stream as a point set dynamically changing in time.

Since in many applications it is not possible to store all point coordinates within local memory, we invented a small data structure to maintain statistics about the dynamically evolving point set. These statistics are maintained in an elaborated fashion, such that certain queries about the point set, for example the question for the average distance between two points, can be answered just using these statistics without knowing the coordinates of each single point. Furthermore our maintained
statistics contain enough information to solve clustering problems on the point set approximately, in
particular the k-means, k-median, and maxcut problem.

Except for the k-median problem, we give the first algorithms in the dynamic geometric streaming model for all considered problems. For the k-median problem we develop the first \((1+\epsilon)\)-approximation algorithm maintaining a small statistic, such that an approximate k-median solution can be constructed using the statistics in reasonable time. The previous algorithm by Indyk was designed to show that in principle a \((1+\epsilon)\)-approximation dynamic streaming algorithm using poly-log space exists [40]. Its running time is \(\tilde{O}(\Delta^{kd}/\epsilon^{d-1})\), such that his algorithm could not be used in practice.

All our algorithms maintain a coreset for the considered problem. The idea is to combine points of the current point set to coreset points. For each coreset point we memorize it’s position and the number of points it represents (called the weight of the coreset point). Since for a small coreset these statistics use much less memory than the information about the whole point set, the idea can be applied to huge data streams, consisting of point sets which do not fit into local memory. Once we have the coreset we can run an arbitrary \((1+\epsilon)\)-approximation algorithm on this small set and quickly get a good approximation for the original point set.

Previously, coresets have been used to design \((1+\epsilon)\)-approximation streaming algorithm for k-median and k-means clustering [37] in the insertion-only case. These algorithms maintain statistics about the distribution of points in exponential grids around the centers of a constant factor approximation. Since the position of these centers can change drastically when points are deleted it seems that one cannot generalize this approach to deletions. Since in this subproject we look at dynamically evolving point sets, we had to invent new techniques to construct coresets in dynamic data streams.

Apart from the specific results, i.e. computing approximations of k-means clustering, k-median clustering, maxcut clustering, and some more problems, the main contribution of the paper is our new method for constructing a coreset. The coreset is obtained by combining statistics about the distribution of points in a certain number of nested grids. To obtain these statistics in the streaming context we use the fact that for a given cost \(Opt\) of the optimal solution the distribution of points within the grids cannot be arbitrary. Most points must lie near the optimal centers and just a few points can be far away from the centers. Using this observation together with other arguments we can prove that essentially statistics about a small number of cells suffice to construct a coreset. This way we can get statistics with higher precision than it would be possible for arbitrary input distributions.

Our algorithms also have applications in a distributed scenario. Assume the input data is distributed over a number of data streams and different computers are monitoring each separate data stream. When we want to answer queries about the union of all point sets seen by all computers, we can easily combine the maintained statistics, compute a single coreset, and compute a solution on this coreset. This solution will be a \((1+\epsilon)\)-approximation of the whole original point set seen in all data streams.

**Sampling in Dynamic Geometric Data Streams and Applications**

We provide randomized algorithms for two well-studied geometric problems over dynamic geometric data streams [33]:

**Maintaining an \(\epsilon\)-approximation of the dynamically evolving point set \(P\).**

Passing over the data stream and it’s insert and delete operations we maintain a small data structure consisting of a small sample of the current point set \(P\). Using this data structure at the end of the stream we can answer certain questions about the current point set like “How many points lie in a rectangle at position \((x, y)\) having side lengths \(a\) and \(b\)?” We can calculate a \((1+\epsilon)\)-approximation on the number of points within the rectangle, even we did not now what rectangle we are asked for at the end of the stream. The approach works for arbitrary ranges of bounded VC dimension as well. Our algorithm uses \(O(\log^3 \Delta \cdot \log^3(1/\delta) \cdot \log(1/\epsilon)/\epsilon^2)\) space, where \(\Delta\) is akin to the spread of the
points in $P$ and is correct with probability $1 - 1/\delta$. The approximations have applications to many other problems, including Tukey depth, simplicial depth, regression depth, the Thiel-Sen estimator, and the least median of squares [21].

Maintaining a $(1 + \epsilon)$-approximation of the cost of minimum weight tree spanning the points in $P$.

Since in many cases the minimum spanning tree is a good network communication structure for wireless communication of sensors at specified positions, the algorithm could be used to get information about optimal communication structures among these sensors without storing all sensor positions. The maintained quantity in turn enables to achieve approximation for other problems, such as TSP or Steiner tree cost. Our algorithms use space $O((\log(1/\delta)) \cdot (\log \Delta/\epsilon)^{O(d)})$, and is correct with probability $1 - \delta$.

The above results are obtained by using a subroutine ”Dynamic Sampling”, which does the following. Consider a sequence of insertions and deletions of elements of a finite Universe $[U] = \{0, \ldots, U - 1\}$. In the geometric setting $[U]$ corresponds to the set of all possible point coordinates, but it could as well be a set of characters in text streams or the set of possible IP addresses in TCP/IP streams. Each element can be inserted as well as deleted multiple times. The subroutine maintains a random element, chosen (almost) uniformly at random from $P$. The difficulty lies in the requirement to choose a point which appears once in the stream with the same probability as a point which appears a thousand times. Just choosing a random element out of the stream would lead to a high probability to choose a point which appears many times. In contrast our method chooses each point of the current point set with the same probability, no matter how often it appears within the data stream. It is immediate that such a procedure yields an $\epsilon$-approximation of desired size.

To compute the weight of the Euclidean minimum spanning tree the above sampling procedure is used in a more subtle way. It is known that the EMST weight can be expressed as a formula depending on the number of connected components in certain subgraphs of the complete Euclidean graph of the current point set [27, 28]. We use an algorithm from [27] to count the number of connected components in these subgraphs. We show that it suffices to maintain a random sample point and all points in a certain radius around this sample point. This task can be also approximately performed by a variant of our sampling routine.

2.4 Algorithms for the analysis and visualization of evolving

2.4.1 State of the art

Dynamic networks naturally occur in many applications such as transport optimization, provision and maintenance of technical services or management of global communication. In most cases a dynamic network is given by a sequence of networks. Each network is a snapshot of an ongoing evolution. Such a snapshot contains all elements present at a given time. Consider the example of the Internet, where packages have to be routed through physical links. There, changes can alter the cost, the technical availability or the commercial usability of edges and nodes.

Various paradigms exist for the graphical representation of dynamic networks. Each approach focuses on one of the following three aspects. First, single snapshots analyse individual properties of a single network at a specific time and track these properties over time. Second, cumulative views combine several snapshots into a static network. This newly created network is restricted to the frequently occurring parts. As a consequence, the contained noise is reduced, thus emphasizing stable characteristics. Third, sequential views target changes over time and track instabilities of the observed evolution. All three aspects implicitly rely on information about esthetic and meaningful visualizations for the corresponding static networks.

Many analyses and applications involve only specific parts of dynamic networks. These are usually
constructed or extracted with help of clustering techniques. Visualizations can, for instance, either reveal dependencies between interacting clusters or focus on the detailed structure of individual clusters.

2.4.2 Main Contributions

Static Visualizations  Analytic visualizations of static networks have been our main objective. Such layouts are based on the results of an analysis while still meeting esthetic criteria. These graphical representations improve the human perception of the analytic result. In many cases, this also exposes additional characteristics that have previously been neglected by the analysis.

Our technique [62] has primarily been evaluated on the Autonomous System (AS) network, which is an abstracted version of the physical Internet. Besides offering a simple graphical representation, the layout has been used to compare real AS networks and artificially generated ones that simulate the AS topology. A standard generator was used to obtain the artificial instances. However, the layout of real and simulated instances strongly differed. Thus the visualization can be used as an additional test for AS network models. An example showing such a comparison is given Figure 5. The layout is based on a graphtheoretical, hierarchical decomposition of the node set.

Another test case was Amazon’s recommendation system. The online shop Amazon (www.amazon.com) offers for each product a list of similar items that have commonly been purchased together with the corresponding product. This list usually contains four selected items, thus the recommendation property is asymmetric. The induced link structure of this network and the WWW share several characteristics. The first property is the small/bounded out-degree but a potentially large in-degree of a node and a power-law like degree distribution. The second property is the fact that products/sites can be structurally important due to either general popularity or to topic specific relevance. Similar to the WWW, the recommendation network as a whole is not available from Amazon, but can be crawled from their webpages.

An example is given in Figure 6, which was crawled in 2003 starting from a car specific topic. Figure 6(a) shows the spatially close neighborhood of the initial product in our layout while the boxes mark different topics. The other two big groups discuss maintenance and restoration of cars. Our layout of the complete crawl is given in Figure 6(b). Colors indicate the relevance of nodes, i.e., red and green symbolize high and low relevance, respectively. Besides the car related topics, other groups comprise novels and guidebooks. The appearance of these products is due to the depth of our search as well as their general popularity.

The third major test case were collaboration networks, i.e., nodes represent people and edges connect nodes if there exist common publications. These networks can be accessed through various sources, for example DBLP (http://www.informatik.uni-trier.de/~ley/db/) is a well maintained database of publications in the computer science field.

An example of such a network is given in Figure 7. The five heads of the departments of the MPII Saarbrücken were chosen as initial set and the network consists of them and their (direct) coauthors. The rank of such a coauthor was determined by the number of collaborations with members of the initial set. Our layout clearly distinguishes between persons with different rank. Moreover the global shape of the network is revealed.

Figure 8 shows another example, where the layers represent the evolution over time. The elevation of a node is determined by the earliest collaboration with (at least) one member of the predetermined initial set. The nodes with the oldest collaborations constitute the bottom layer. Again, our layout uncovers the global structure of the network. Similar to the previous examples, groups can be clearly identified and their interaction can be studied thanks to the high readability of the layout. In contrast to them, the visualized information in Figure 8 exhibits a temporal evolution. Although the representation is static, major evolutionary steps can be observed.
Figure 5: Comparison of real and simulated AS network (colors correspond to importance in the hierarchy).
Figure 6: A part of Amazon's recommendation network. It was crawled in 2003 starting with a car specific topic.

Figure 7: The collaborations of the five heads of the departments of the MPII Saarbrücken
Dynamic Visualizations  Based on our experience with static visualization, we are adapting previous results as to allow for sequential views. Traditional sequential views are not suitable for large networks due to the lack of visible details. We currently research different graphical means to cope with such networks. A promising approach stacks layouts of successive snapshots transparently on top of each other. This technique is very similar to our static layouts, where importance defined the height of an element and not time. We are still able to code the importance with help of other graphical features such as shape, size, or color. In addition, this method can be extended to the edges in a continuous way (Figure 9) that improves human perception. The adaption of the node-edge incidences is achieved by applying a soft-focus effect. As a convenient network type, we are using collaboration networks. These are readily available and usually well maintained.

The layouts in Figure 10 primarily focus on temporal evolution. Each layer corresponds to a fixed point in time, while nodes and edges represent coworkers and collaborations, respectively. Edges between different layers connect identical nodes through time. The size of a node and the thickness of an edge both represent the number of publications. Associated analyses can be performed on various scales, e.g., on a microscopic scale, which emphasizes details, and on a macroscopic scale revealing major trends. In Figure 10(a), we can observe a change of collaboration (the four big nodes in the center), while Figure 10(b) reveals the development of node relevance, since modifications of the position and size of nodes are easily traceable.

Application Specific Analyses  In order to obtain meaningful static visualizations, the inherent structure of a network has to be explored. Therefore, besides researching visualization techniques, we
studied algorithmic characteristics of various network properties, for example, fast computability of clustering coefficients [66]. In the case of the Autonomous System network, we studied the commercial relationships modeled by the edges [63] and general reduction techniques [64]. The insight we gained substantially improved our static layout technique. We present a case where the visualization not only improves the analysis but also (algorithmic) computations in [67]. We were able to accelerate the computation of shortest paths using a suitable layout. Other techniques to speed up the computation are based on network decomposition [65].

3 Summary

In the implementation plan we proposed to mine the inner structure of large self-organizing evolving networks based on the Web, to design data stream and random sampling algorithms in order to maintain topological and statistical properties of dynamic networks, to visualize dynamic networks at the level of clusters.

We followed the implementation plan. First of all, our work suggests a refinement of the bow-tie pictorial view of the Web graph [1]. The bow-tie picture seems too coarse to describe the details of the Web. The picture that emerges from our work can better be described by the shape of a daisy: the IN and OUT regions are fragmented into a large number of small and shallow petals (the WCCs) hanging from the central dense CORE.

As a second point, we studied which problems could be solved using data stream algorithms [7, 32, 33]. As start point, we considered an algorithm proposed in the literature, modified it (using a different family of hash functions), and tested in Web graphs. Moreover we developed a general technique for solving a number of fundamental problems over dynamic geometric data streams approximately.

Finally we developed an integral approach for static visualizations applicable to various types of networks [62]. These layouts are very flexible and can incorporate divers analytic properties. In addition to respecting esthetic criteria such as crossing minimization and compactness, the readability
and the human perception of the included analytic characteristics are improved. Thus, the exploration of networks can enhance an analysis by means of the visualization techniques we presented. For example, our layout can be considered as an additional evaluation for AS topology generators.

Based on these fundamental results, we explore different methods to transfer our knowledge of the static case to the dynamic case. First observations suggest that temporal information can be included in a limited way in single snapshots. Further studies show that small modifications of our original framework already lead to feasible visualizations of dynamic networks. At the same time, however, we identified problematic patterns that are currently not fully resolved, such as futural dependencies.

Moreover we plan to develop and implement data stream algorithms for counting minors in large graphs. The minors of main interest are triangles and small bipartite cliques. Currently we are working on sampling techniques for computing triangles and bipartite cliques of size three. Counting the number of triangles of a graph is a fundamental problem in network analysis, e.g., it can be used to compute the clustering coefficient of a network. When considering the graph formed by the hyperlinked structure of the web, the clustering coefficient gives an indication of the presence of communities, which is an important measure performed in such kind of graphs. Likewise, small bipartite cliques are the core of such communities.

Finally, we are going to collect larger, and more “realistic” crawls, and perform the same measurements to verify our hypothesis on the inner structure of the Web graphs. Our current results are sensitive to the choices and limitations of the crawlers, and it is not clear if the available crawls are representative of the actual Web graph. Unfortunately till now, there are no publicly available crawls that have been collected with the aim of validating our hypothesis on the structure of the Web graph.

References


